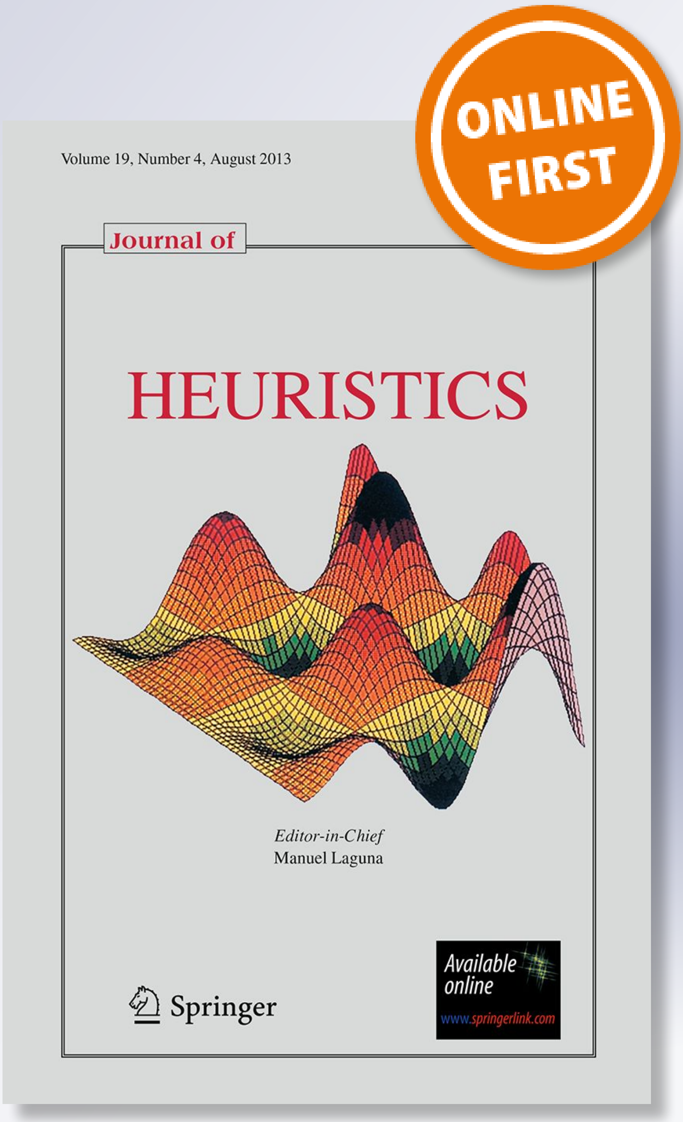
Estimating searching cost of regular path queries on large graphs by exploiting unit-subqueries

Van-Quyet Nguyen, Quyet-Thang Huynh & Kyungbaek Kim

Journal of Heuristics

ISSN 1381-1231

J Heuristics
DOI 10.1007/s10732-018-9402-0



Journal of Heuristics
https://doi.org/10.1007/s10732-018-9402-0

Estimating searching cost of regular path queries on large graphs by exploiting unit-subqueries

Van-Quyet Nguyen[1] · Quyet-Thang Huynh[2] · Kyungbaek Kim[1]

Received: 13 March 2018 / Revised: 29 September 2018 / Accepted: 24 November 2018

Abstract
Regular path queries (RPQs) are widely used on a graph whose answer is a set of tuples of nodes connected by paths corresponding to a given regular expression. Traditional automata-based approach for evaluating RPQs is restricted in the explosion of graph size, which makes graph searching take high cost (i.e. memory space and response time). Recently, a cost-based optimization technique using rare labels has been proved to be effective when it is applied to large graph. However, there is still a room for improvement, because the rare labels in the graph and/or the query are coarse information which could not guarantee the minimum searching cost all the time. This is our motivation to find a new approach using fine-grained information to estimate correctly the searching cost, which helps improving the performance of RPQs evaluation. For example, by using estimated searching cost, we can decompose an RPQ into small subqueries or separate multiple RPQs into small batch of queries in an efficient way for parallelism evaluation. In this paper, we present a novel approach for estimating the searching cost of RPQs on large graphs with cost functions based on the combinations of the searching cost of unit-subqueries (i.e. every smallest possible query). We extensively evaluated our method on real-world datasets including Alibaba, Yago, Freebase as well as synthetic datasets. Experimental results show that our estimation method obtains high accuracy which is approximately 87% on average. Moreover, two comparisons with automata-based and rare label based approaches demonstrate that our approach outperforms traditional ones.

Keywords Regular path queries · Unit-subqueries · USCM · Graph querying

✉ Kyungbaek Kim
kyungbaekkim@jnu.ac.kr

Van-Quyet Nguyen
quyetict@utehy.edu.vn

Quyet-Thang Huynh
thanghq@soict.hust.edu.vn

1 Chonnam National University, 77, Yongbong-ro, Buk-gu, Gwangju, South Korea

2 Hanoi University of Science and Technology, 01, Dai Co Viet Road, Hanoi, Vietnam

Published online: 30 November 2018

1 Introduction

A regular path query (RPQ) is introduced as a part of a query language for graph databases, which are represented as graphs whose nodes are objects and edge labels specify relationships between them (Mendelzon and Wood 1995). The answer of an RPQ is a set of tuples of nodes which are connected with edge labels in some ways by the paths specified by a regular language (Barceló et al. 2012; Calvanese et al. 1999; Consens and Mendelzon 1990; Cruz et al. 1987; Libkin and Vrgoč 2012). RPQs have been utilized in many applications such as friends recommendations in social networks (Konstas et al. 2009) and detecting signal pathways in protein interaction networks (Scott et al. 2006). In such systems, databases could store extremely large graphs in practice (i.e. hundreds of millions nodes and edges on Twitter social network (Yang and Leskovec 2011), billions nodes/edges on Friendster social network (Yang and Leskovec 2015)). Hence, evaluating an RPQ on such graphs takes high cost causing substantial memory spaces and long response time.

A common approach for evaluating of RPQs is to use automata (Goldman and Widom 1997). However, the drawback of the automata-based approach is that the states of automaton are mapped onto the graph, which could cause long response time in case of large graphs. To address this issue, there have been several studies focusing on optimizing the searching cost of RPQs. The first technique is rewriting regular path queries (Calvanese et al. 1999; Fernandez and Suciu 1998). In which, a given regular expression is converted into another one that helps reducing search space by searching only a portion of the data. But, this approach still has a limitation when dealing with rewriting highly complex RPQs (e.g. nested RPQs with modifier recursion).

In recent years, a cost-based optimization technique using rare labels for RPQs evaluation has been proved to be effective when it is applied to large graphs (Koschmieder and Leser 2012). The authors used a cost-based technique for determining which labels in the graph are considered to be rare, then they are used to decrease the search space. However, the major drawback of this approach is that the algorithm depends on the presence of rare labels and the number of rare labels in the graph and query. Therefore, this technique could not guarantee the minimum searching cost all the time. There is a room for improvement, in which estimating the searching cost of RPQs will open the way to improve the performance of query evaluation. For instance, to reduce the response time for evaluating an RPQ, we can split original RPQ into smaller subqueries, evaluate them parallelly and combine partial answers. In this case, estimating the searching cost of each subquery is one of the key points to choose the split labels which help to separate original RPQ in an efficient way. Such reasoning is the basis of the approach we propose in this paper.

In this paper, we present a novel approach for estimating the searching cost of RPQs on large graphs with cost functions based on the combinations of the searching cost of unit-subqueries (i.e. every smallest possible query). In our method, we exploit unit-subqueries to make a so-called USCM (Unit-Subquery Cost Matrix), which presents the searching cost of the unit-subqueries. We provide cost functions based on USCM to estimate the searching cost of an RPQ by decomposing the original query into a set of unit-subqueries.

Our work makes the following contributions.

- We define a USCM (Unit-Subquery Cost Matrix) according to the searching cost of unit-subqueries. A unit-subquery can be a part of an RPQ, so we can use USCM to estimate the searching cost of the RPQs.
- According to USCM, we propose cost functions for estimating the searching cost of a given RPQ. Three main operators in an RPQ including concatenation, alternation, and bounded Kleene operator, are considered to estimate the cost. Moreover, we also discuss the estimation of highly complex RPQs.
- We present how our idea can be applied for improving the performance of RPQs evaluation by reducing the searching cost in two applications: parallel evaluation and multi-query optimization of RPQs.
- We conduct extensive experiments which show that our estimation method obtains high accuracy which is approximately 87% on average. Moreover, two comparisons with automata-based and rare label based approaches demonstrate experimentally that our approach outperforms traditional ones in the aspect of parallel RPQs evaluation.

The rest of this paper is organized as follows. Section 2 presents an overview of related works. In Sect. 3, we present terms and definitions related to regular path queries. Section 4 describes our method of estimating the searching cost of RPQs: a Unit-Subquery Cost Matrix (USCM) and estimating the searching cost of RPQs by using USCM. Section 5 discusses how our proposed approach can be applied for improving the performance of RPQs evaluation. We conduct the experimental evaluation using both real-world and synthetic graphs in Sect. 6. Section 7 concludes with a summary and shows our future work.

2 Related work

Regular path queries evaluation on graph database has been studied intensively in the literature (Barceló Baeza 2013; Goldman and Widom 1997; Grahne and Thomo 2000; Koschmieder and Leser 2012; Libkin and Vrgoč 2012; Trißl 2007; Yakovets et al. 2016). The most common approach for evaluating an RPQ is based on automata. A graph needs to be translated into an NFA (Nondeterministic Finite Automaton), and a regular expression of an RPQ can be converted into an automaton before being used it to match paths (Goldman and Widom 1997). However, the drawback of the automata-based approach is that the states of automaton are mapped onto the graph, which could cause long response time. Because of that, there have been several studies focusing on optimization the searching cost of RPQs underlying automata-based.

A common strategy for reducing the searching cost is query optimization. The first technique is rewriting regular path queries (Calvanese et al. 1999; Fernandez and Suciu 1998). Fernandez and Suciu (1998) presented two optimization techniques based on graph schemas: (1) query pruning which used to rewrite a given regular path expression into another one that helps reducing search space by searching only a portion of the data; and (2) query rewriting using state extent which rewrites the query into the one that starts traveling data from entry points deeper in the graph, instead of from the root which can avoid all navigation. Calvanese et al. (1999) proposed a view-based query rewriting for RPQs in semi-structured data which guarantees to provide only answers contained in those of the original query. Other rewriting approaches for optimizing regular path queries are presented by Grahne and Thomo (2003). However, the query rewriting techniques still have some limitations deal with highly complex RPQs (e.g. nested RPQs with modifier recursion), which leads to state explosion after converting the rewritten query to Deterministic Finite Automata (DFA) for graph searching. Therefore, there have been several techniques also proposed for estimating query size (Liu et al. 2014) or minimizing DFAs (Almeida and Zeitoun 2008; Liu et al. 2016).

Recently, cost-based optimizations of RPQs is proved to be effective to deal with a large graph. Koschmieder and Leser (2012) used a cost-based technique for determining which labels to be considered rare. By using rare labels as start-, end-, and way-points during traversal, this approach decreases the search space and get a more efficient approach to the RPQ evaluation. However, the drawback of this approach is that the algorithm depends on the presence of rare labels. In the case of poor rare labels on the graph and the RPQs, or long queries, this approach still takes a high cost which the complexity can reach $\mathrm{O}(n^2)$, where n is the number of edges of the graph. We will compare our proposed approach to rare label based approach in case of parallel RPQs evaluation (for more details, see Sect. 6).

An area, where the efficiency of evaluation RPQs is important, is querying on distributed graphs. A survey about the state of the art of evaluating queries on distributed graphs is presented by Kossmann (2000). Partial evaluation technique is used to evaluate XPath queries on distributed XML data modeled as trees (Cong et al. 2007; Le Anh and Kiss 2007). Suciu (2002) presented a distributed query evaluation approach on semi-structured data. Their algorithm takes a bounded complexity $O(n^2)$ for the amount of data transferring via the network, where n is the total of cross-edges. Fan et al. (2012) proposed efficient algorithms for answering three classes of regular reachability queries on distributed graphs based on a technique named *partial evaluation*. However, it faces a communication bottleneck problem when assembling all distributed partial query results. This problem is addressed in studies which are proposed by Nguyen-Van et al. (2013) and Tung et al. (2013); therein, a large amount of redundant data is detected and removed before assembling at the coordinate site.

Despite many studies have focused on RPQs evaluation, to the best of our knowledge, there have been only a few researches studying on estimating the searching cost of RPQs and its effectiveness. Trißl and Leser (2010) provided functions to estimate the sizes of result sets and the response times to evaluate reachability and path queries. Davoust and Esfandiari (2016) presented estimation cost functions to provide strategies for evaluating RPQs on distributed graphs. However, this work mainly focuses on estimating the amount of data to be transferred via the network during evaluating an RPQ. None of these works above provides estimation cost functions relying on operators in RPQs and connectivity of labels. In this paper, we propose a novel approach for estimating the searching cost of RPQs on large graphs.

3 Preliminaries

3.1 Graph data and regular path queries

We consider an edge-labeled directed graph $G = (V, E, \Sigma)$, where V is a finite set of nodes, Σ is a finite set of labels, and $E \subseteq V \times \Sigma \times V$ is a finite set of edges. An edge (v, a, u) denotes a directed edge from node v to u labeled with $a \in \Sigma$.

A path ρ between nodes v_0 and v_k in G is a sequence

$$\rho = v_0 a_0 v_1 a_1 v_2 \dots v_{k-1} a_{k-1} v_k$$

such that each (v_i, a_i, v_{i+1}), for $0 \leq i < k$, is an edge. The sequence of labels of a path ρ, denoted L(ρ), is the string $a_0 a_1 \dots a_{k-1} \in \Sigma^*$, where Σ^* is a set of all possible strings over the set of labels Σ. We also define the *empty* path as (v, ε, v) for each $v \in V$; the label of such a path is the empty string ϵ.

An RPQ with a regular expression R is a query of the form $Q(R) = v \xrightarrow{L(R)} u$, where $L(R) \in \Sigma^*$ is a regular language. So, a path ρ satisfies $Q(R)$ on the graph G iff L(ρ) $\in$ L(R), then ρ is an answer of $Q(R)$. Here, R is a regular expression over Σ,

$$R = \epsilon \mid a \mid R \circ R \mid R \cup R \mid R^{[i,j]},$$

where ϵ is an empty value; a is a label in Σ; $R \circ R$, $R \cup R$, and $R^{[i,j]}$ denote concatenation, alternation, and Kleene operator with bounded recursion $[i, j]$, where $i < j$ and $i, j \in N$, respectively. Note that, in the syntax of regular expression we use a bounded Kleene operator which bounds recursion with $[i, j]$ instead of an unbounded Kleene operator (e.g., *, +). This is motivated by the following three observations. Firstly, the bounded Kleene operator is supported by graph query languages in practice, such as Neo4j's Cypher.[1] Secondly, bounded recursion on regular path queries evaluation has been studied in the literature (Fletcher et al. 2016). Finally, it is not difficult to find that for any graph G, there exists a natural number n such that for every RPQ, $Q(R)$, there is a possible case that $R^* = R^{[0,n]}$. Similarly, other modifiers (+ and ?) can be represented as follows: $R+ = R^{[1,n]}$ and $R? = R^{[0,1]}$. While evaluating an RPQ with unbounded recursion on a large graph is a non-trivial task.

Example 1 Figure 1a illustrates a graph G of a social network, where each node denotes a person with his/her name and each edge represents the relationship between two people, in which an edge is labeled by an element in a set of labels

$$\Sigma = \{supervisor, colleague, friend, married, knows\}.$$

In this graph, a regular path query $Q(R)$ with

– $R = suppervisor \circ (colleague \cup friend)$ finds all paths between any $supervisor(s)$ and $colleagues$ or $friends$ of his/her employees. In this case,

1 http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/.

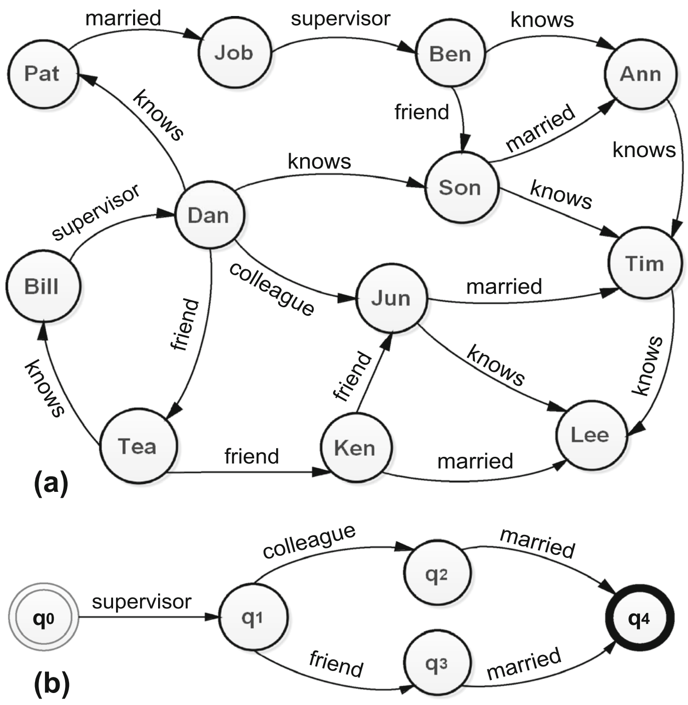


Fig. 1 **a** An example of a social network as a directed edge-labeled graph; **b** a simple regular path query as an automaton

the result includes three paths as follows.

$$\text{Bill} \xrightarrow{supervisor} \text{Dan} \xrightarrow{colleague} \text{Jun}$$
$$\text{Bill} \xrightarrow{supervisor} \text{Dan} \xrightarrow{friend} \text{Tea}$$
$$\text{Job} \xrightarrow{supervisor} \text{Ben} \xrightarrow{friend} \text{Son}$$

– $R = supervisor \circ friend^{[1,2]} \circ married$ finds all paths from any $supervisor(s)$ to the people who married with $friend$ or $friend$ of $friend$ of his/her employees. In this case, the result includes two paths as follows.

$$\text{Bill} \xrightarrow{supervisor} \text{Dan} \xrightarrow{friend} \text{Tea} \xrightarrow{friend} \text{Ken} \xrightarrow{married} \text{Lee}$$
$$\text{Job} \xrightarrow{supervisor} \text{Ben} \xrightarrow{friend} \text{Son} \xrightarrow{married} \text{Ann}$$

3.2 Evaluation of an RPQ

Informally, the evaluation of an RPQ, $Q(R)$, is to find all paths between pairs of nodes in a graph G, such that the path from one node to the other matches a given regular expression R. There are two types of RPQs which have been considered in the literature. In which, *multi-source queries* start a search at every node in the graph, and *single-source queries* start a search at a single given *start node*. In this paper, we consider estimating the searching cost of the multi-source queries. For better understanding of the estimation cost, we will describe the basic of RPQ evaluation using automata-based technique.

Query automaton To process an RPQ, a regular expression can be converted into an automaton then used to matching paths. We use a deterministic finite automata (DFA) to represent query where the definition of DFA is like the study presented by Hopcroft et al. (2006). That is, an RPQ, $Q(R)$, is represented by an automaton A_R which is a 5-tuple as the following:

$$A_R = \{Q, \Sigma, \mu, q_0, F\},$$

where Q is a finite set of states, Σ is a finite set of labels (or symbols), μ is the transition function, that is, μ: $Q \times \Sigma \rightarrow Q$, q_0 is an initial (or start) state and $q_0 \in Q$, F is a set of terminal states and $F \subset Q$.

Query evaluation A well-known method for query evaluation (Hopcroft et al. 2006) based on automata consists of the steps as follows:

- Build a finite automaton A_R associated with the regular expression R. The initial state of A_R is q_0, the accepted states are $\{q_t\}$, where $q_t \in F$.
- Consider graph G as an automaton A_G with nodes as states, edges as transitions and compute the cross-product of the automata $A_P = A_R \times A_G$.
- Apply any graph search algorithms such as breadth-first or depth-first to find all pairs of nodes related by the regular path: search A_P from all initial states (q_0, v_i) to find all reachable accepted states (q_t, v_j). All pairs of nodes (v_i, v_j) are answers to the RPQ.

The evaluation cost of the algorithm above consists of the cost of building the query automaton, plus the cost of building and searching the product automaton. In practice, the searching cost is the determinant of evaluation cost, especially in the case of handling large graphs. We define the searching cost of an RPQ as the number of traversed edges for searching paths corresponding to the RPQ.

Example 2 Suppose that we have a graph G as described in Example 1. We consider the evaluation of an RPQ, $Q(R)$, with

$$R = supervisor \circ (colleague \cup friend) \circ married.$$

First, we convert $Q(R)$ into finite automata A_R as shown in Fig. 1b. Next, we look up the *start nodes* from graph G. To be able to efficiently gather *start nodes* from the graph, we assume that an index of the edge label also encodes their incoming and outgoing nodes. This index gives us the list of *start nodes* of the label in the graph. In this case, we have two *start nodes* $\{Bill, Job\}$. Starting from node $Bill$, there is a node, Dan, matching with a state in A_R (q_1). From Dan, we find the nodes for next searching. Here, four edges are traversed and two nodes, $\{Jun, Tea\}$, are considered as the next starting nodes. Then, the searching process continues from these nodes, there are four more edges be traversed and only one node, Tim, is matched with the final state q_4. Thus, for searching from $Bill$, the searching cost is nine. Similarly, starting from Job, we have only one path satisfying $Q(R)$,

$$\text{Job} \xrightarrow{supervisor} \text{Ben} \xrightarrow{friend} \text{Son} \xrightarrow{married} \text{Ann},$$

with the searching cost is five. As the result, we have the searching cost of Q(R) in this case is fourteen. We are going to compare this the result with our estimation cost in the next section.

4 Estimating the searching cost of RPQs

In this section, we present a novel approach for estimating the searching cost of RPQs. We first define a Unit-Subquery Cost Matrix (USCM). We then present how to estimate the searching cost by using USCM.

4.1 Unit-Subquery Cost Matrix (USCM)

Intuitively, an RPQ is composed of multiple small subqueries with a few operators such as concatenation, alternation, and bounded Kleene. Then, we can define a *unit-subquery* as the smallest subquery which is concatenated by two labels from Σ; the *start label* and the *end label*. For example, in the graph G of Fig. 1, a subquery $Q(supervisor \circ colleague)$ is a unit-subquery, where $supervisor$ is the *start label* and $colleague$ is the *end label*. In practice, any query which is defined as in Sect. 3.1 can be split into multiple unit-subqueries even if the query which contains the Kleene operators with bounded recursion. For example, the subquery $Q((colleague \cup friend) \circ married)$ can be split into two unit-subqueries $Q(colleague \circ married)$ and $Q(friend \circ married)$; meanwhile, the query $Q(supervisor \circ (colleague \circ friend)^{[1,2]})$ is composed of six unit-subqueries including $Q(supervisor \circ colleague)$, $Q(supervisor \circ friend)$, $Q(colleague \circ friend)$, $Q(colleague \circ colleague)$, $Q(friend \circ colleague)$, and $Q(friend \circ friend)$.

The cost of a unit-subquery is defined as the number of edges with the *end label*, which is connected to the edges with the *start label*. For example, in the graph G of Fig. 1, the cost of a subquery $Q(supervisor \circ colleague)$ is one because there is only one edge (Dan, $colleague$, Jun) labeled with $colleague$, which is connected to uni-directional edges labeled with $supervisor$.

With the definition of the cost of unit-subqueries, we can generate a *Unit-Subquery Cost Matrix (USCM)* which represents the cost of all possible unit-subqueries from Σ. An example of USCM is shown in Table 1. The size of USCM is n by $n+1$ where n is the number of distinct labels in Σ. A cell (i, j) of USCM, except the last column where j is $n+1$, represents *the cost of a unit-subquery*, $Q(a_i a_j)$, whose *start label* is $a_i \in \Sigma$ and *end label* is a_j. For clarity of presentation, we drop the explicit use of the concatenation, and we use the symbol | for alternation operator in terms and equations, only keep the symbols $\circ$ and $\cup$ in the examples (from now on). In the last column,

Table 1 An example of Unit-Subquery Cost Matrix

Label: count	*supervisor*	*colleague*	*friend*	*married*	*knows*	Total
supervisor: 2	0	1	2	0	3	6
colleague: 1	0	0	0	1	1	2
friend: 4	0	0	2	3	3	8
married: 4	1	0	0	0	2	3
knows: 8	1	0	0	2	4	7

a cell (i, j) represents the cost of a unit-subquery $Q(a_i\_)$, that is, the summation of the costs of unit-subqueries whose *start label* is a_i. Additionally, USCM contains the number of edges with a given label (Count) like the first column of USCM.

Because the contents of USCM are constant in a given graph G, we can prepare USCM for just one time unless the graph G is updated. The complexity of building USCM is $n \times (|E| + |E|)$.

4.2 USCM-based estimating the searching cost

In this section, we propose cost functions for estimating the searching cost of an RPQ, $Q(R)$, in three main cases of regular expression R: (1) a simple regular expression with concatenation operator; (2) a regular expression with alternation operator; and (3) a regular expression with bounded Kleene operator. We also discuss estimating the searching cost of highly complex RPQs.

4.2.1 An RPQ with concatenation

In our approach, the searching cost of an RPQ is estimated by splitting the original RPQ into multiple unit-subqueries and gathering the cost of each successive unit-subqueries.

Let us assume that there is an RPQ, $Q(R)$, where $R = a_0a_1 \ldots a_n$ as a string that is concatenated by $(n + 1)$ labels $a_i \in \Sigma$. Then, $Q(R)$ can be split into $Q(a_0a_1), Q(a_1a_2), \ldots, Q(a_{n-1}a_n)$, and the searching cost for $Q(R)$ is defined as summation of cost of each successive unit-subquery like Eq. 1.

$$C_{Q(R)} = \sum_{i=0}^{n-1} C_{Q(a_ia_{i+1})} = \sum_{i=0}^{n-1} C_i \tag{1}$$

For C_0, the evaluation starts from the edge labeled with a_0 and tries to find the path to a_1. Accordingly, C_0 composed of the cost of finding the next search nodes and the cost of searching a_1. That is, $C_0 = \delta(a_0) + \xi(a_0)$, where $\delta(a_i)$ is the number of edges given label a_i which is the *Count* value for the first column of USCM and $\xi(a_i)$ is the cost of $Q(a_i\_)$ which is the value of the last column of USCM.

For C_i, where $i > 0$, we do not consider the searching cost for finding edges with label a_i, because this cost is already considered in the previous step C_{i-1}. So, for

C_i, we only consider the searching cost for finding edges with label a_{i+1}. However, this cost is affected by the number of search nodes which are found in the previous step(s). To consider this effect, we can calculate the probability of how many edges labeled with a_i related to the unit subquery $Q(a_{i-1}a_i)$ are found among the all the edges labeled with a_i, and apply to the searching cost to edges labeled with a_{i+1} from edges labeled with a_i. That is, C_i can be represented like Eq. 2, where $\mu(a_{i-1}, a_i)$ is the cost of unit-subquery $Q(a_{i-1}a_i)$, which is the first value of each cell of USCM.

$$C_i = \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{i-1}, a_i)}{\delta(a_i)} \times \xi(a_i) \tag{2}$$

Example 3 Suppose that we have a graph G as described in Example 1. An actual situation as the following: for a marketing strategy in a company, the board of directors wants to introduce products through all paths from the supervisors to people who are married to friends of the employees in that company. A regular expression R which represents that situation is $R = supervisor \circ friend \circ married$. Here, we do not focus on evaluating this query but estimate the searching cost of $Q(R)$. In this case, the searching cost can be estimated as follows.

$$\begin{aligned} C_0 &= C_{Q(supervisor \circ friend)} \\ &= \delta(supervisor) + \xi(supervisor) = 2 + 6 = 8 \\ C_1 &= C_{Q(friend \circ married)} \\ &= \frac{\mu(supervisor, friend)}{\delta(friend)} \times \xi(friend) = 2/4 \times 8 = 4. \end{aligned}$$

Thus, the total estimated cost is twelve. It is equal to the true cost of evaluating $Q(R)$ in the graph G by using automata-based approach.

4.2.2 An RPQ with alternation operator

We assume that an RPQ, $Q(R)$, is defined by a regular expression,

$$R = a_0 \ldots a_{i-1}(a_i|a_{i+1})a_{i+2} \ldots a_n,$$

where $a_i \in \Sigma$. Herein, R has an alternation operator between a_i and a_{i+1}. In this case, the original $Q(R)$ can be split into three subqueries $Q(a_0 \ldots a_{i-1})$, $Q(a_{i-1}(a_i|a_{i+1}) a_{i+2})$, and $Q(a_{i+2} \ldots a_n)$. For the subqueries $Q(a_0 \ldots a_{i-1})$ and $Q(a_{i+2} \ldots a_n)$, we can estimate their cost by using our method in Sect. 4.2.1.

For evaluating $Q(a_{i-1}(a_i|a_{i+1})a_{i+2})$, we need to consider two different steps: $Q(a_{i-1}(a_i|a_{i+1}))$ and $Q((a_i|a_{i+1})a_{i+2})$. In the first step, $Q(a_{i-1}(a_i|a_{i+1}))$ can be considered by two subqueries $Q(a_{i-1}a_i)$ and $Q(a_{i-1}a_{i+1})$. Here, evaluating both of these subqueries starts from edges labeled with a_{i-1}, and during a single evaluation time, we can traverse the edges labeled with a_i as well as a_{i+1}. So, the cost of $Q(a_{i-1}(a_i|a_{i+1}))$ can be estimated by the cost of either $Q(a_{i-1}a_i)$ or $Q(a_{i-1}a_{i+1})$. On the other hands, $Q((a_i|a_{i+1})a_{i+2})$ can be decomposed into $Q(a_ia_{i+2})$ and

$Q(a_{i+1}a_{i+2})$, and the evaluation process of these subqueries is different to each other. So, C_A, the estimated cost of $Q(a_{i-1}(a_i|a_{i+1})a_{i+2})$, can be estimated by the summation of the costs of $Q(a_{i-1}a_i)$, $Q(a_ia_{i+2})$, and $Q(a_{i+1}a_{i+2})$ like Eq. 3.

$$C_A = C_{Q(a_{i-1}a_i)} + C_{Q(a_ia_{i+2})} + C_{Q(a_{i+1}a_{i+2})} \tag{3}$$

Equation 3 can be represented as an explicit formula by two cases as the following:

- i = 1

$$C_A = \delta(a_0) + \xi(a_0) + \frac{\mu(a_0, a_1)}{\delta(a_1)}\xi(a_1) + \frac{\mu(a_0, a_2)}{\delta(a_2)}\xi(a_2) \tag{4}$$

- i > 1

$$\begin{aligned} C_A = {} & \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \cdots \times \frac{\mu(a_{i-2}, a_{i-1})}{\delta(a_{i-1})} \\ & \times \left(\xi(a_{i-1}) + \frac{\mu(a_{i-1}, a_i)}{\delta(a_i)}\xi(a_i) + \frac{\mu(a_{i-1}, a_{i+1})}{\delta(a_{i+1})}\xi(a_{i+1})\right) \end{aligned} \tag{5}$$

Example 4 We illustrate our idea of estimating searching cost in case of query has alternation operator by an example. In which, the regular expression $R = supervisor \circ (colleague \cup friend) \circ married$. By using Eq. 4, the searching cost of $Q(R)$ can be estimated as follows.

$$\begin{aligned} C_{Q(R)} = C_A = {} & \delta(supervisor) + \xi(supervisor) \\ & + \frac{\mu(supervisor, colleague)}{\delta(colleague)}\xi(colleague) \\ & + \frac{\mu(supervisor, friend)}{\delta(friend)}\xi(friend) \\ & = 2 + 6 + (1/1) \times 2 + (2/4) \times 8 = 14 \end{aligned}$$

In this case, the total estimated cost is fourteen. It equals the true cost of evaluating $Q(R)$ in the graph G by using automata-based approach as we mentioned in Example 2.

4.2.3 An RPQ with bounded Kleene operator

Let us assume that there is an RPQ, $Q(R)$, where

$$R = a_0a_1 \dots a_{k-1}a_k^{[i,j]}a_{k+1} \dots a_n$$

with a bounded Kleene operator. To estimate the cost of $Q(R)$, we can split this query into three subqueries including

$$Q(a_0 \dots a_{k-1}),\ Q\left(a_{k-1}a_k^{[i,j]}a_{k+1}\right), and\, Q(a_{k+1} \dots a_n),$$

then the searching cost for $Q(R)$ is defined as the summation of the cost of each subquery. We can estimate the costs of the subqueries $Q(a_0 \dots a_{k-1})$ and $Q(a_{k+1} \dots a_n)$ by using the proposed method described in Sect. 4.2.1. The subquery $Q(a_{k-1}a_k^{[i,j]}a_{k+1})$ is composed of the unit-subqueries: a $Q(a_{k-1}a_k)$, $(j-1)$ times $Q(a_k a_k)$, and a $Q(a_k a_{k+1})$. So, the estimated cost of $Q(a_{k-1}a_k^{[i,j]}a_{k+1})$, C_K, is defined as shown in Eq. 6.

$$C_K = \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \dots \times \frac{\mu(a_{k-1}, a_k)}{\delta(a_k)} \left(1 + \frac{\mu(a_k, a_k)}{\delta(a_k)} + \frac{\mu(a_k, a_k)}{\delta(a_k)} \times \frac{\mu(a_k, a_k)}{\delta(a_k)} + \cdots \right) \xi(a_k) \tag{6}$$

Let $\omega = \dfrac{\mu(a_k, a_k)}{\delta(a_k)}$, $\forall \omega \neq 1$, the estimated cost, C_K, can be formalized as follows.

$$C_K = \frac{\mu(a_0, a_1)}{\delta(a_1)} \times \dots \times \frac{\mu(a_{k-1}, a_k)}{\delta(a_k)} \times \frac{\omega^j - 1}{\omega - 1} \xi(a_k) \tag{7}$$

Equation 7 shows that the cost of evaluating an RPQ does not depend on the lower bound (i) of Kleene operator, but depends on the upper bound (j) of Kleene operator. That is, the high value of j will take a high searching cost of $Q(R)$.

Example 5 In this example, we will estimate the searching cost of RPQ, $Q(R)$, with $R = supervisor \circ friend^{[1,3]} \circ married$. In this case, $C_{Q(R)} = C_0 + C_K$, with $C_0 = \delta(supervisor) + \xi(supervisor) = 2 + 6 = 8$; and $C_K = \dfrac{\mu(suppervisor, friend)}{\delta(friend)} \times \dfrac{\omega^3 - 1}{\omega - 1}\xi(friend)$, where $\omega = \dfrac{\mu(friend, friend)}{\delta(friend)} = 2/4 = 0.5$. So, $C_K = 2/4 \times (0.5^3 - 1)/(0.5 - 1)) \times 8 = 7$. Finally, we have $C_{Q(R)} = 8 + 7 = 15$. It is close to true cost, *sixteen*, of evaluating $Q(R)$ on graph G.

4.3 Estimating highly complex RPQs

Our approach is not only effective with simple RPQs but also highly complex RPQs such as the query with a regular expression of the form $R \circ S^{[i,j]} \circ T$ or $R \circ (S^{[i,j]} \circ T)^{[x,y]} \circ U$, where R, S, T, and U are regular expressions, and i, j, x, and y are natural numbers, in which $i < j$ and $x < y$. To estimate the cost of a complex RPQ, we first decompose such query into the queries containing at least one of three cases of operators as described in Sect. 4.2. We then use USCM to estimate the cost for decomposed queries. For example, an RPQ, Q(R), with $R = supervisor \circ (colleague \cup friend)^{[1,2]} \circ married$ can be decomposed into two queries:

(1) $supervisor \circ colleague \circ (colleague \cup friend) \circ married$ and
(2) $supervisor \circ friend \circ (colleague \cup friend) \circ married.$

It is not difficult to estimate the searching cost of these queries by using our proposed as presented above.

5 Applications

Our idea can be used for variations of the general RPQ problem, such as parallel evaluation or multi-query optimization of RPQs. In this section, we show how our proposed method can be applied for improving these two main RPQs problems.

Parallel evaluation of RPQs As we mentioned in the previous section, splitting an RPQ into small subqueries by using rare labels, then evaluate them in parallel and combined partial answers, have been proved to be effective when applying to large graphs (Koschmieder and Leser 2012). However, this technique could not guarantee the minimum searching cost all the time. Our proposed method of estimating the searching cost of RPQ is one of the key points to improve the performance for parallel evaluation of RPQs. To do this, we can follow the steps as below.
Step 1 Find all N possible sets of subqueries, $S = \{S_1, S_2, \ldots, S_N\}$, for a given RPQ. For example, we can find three sets of subqueries for an RPQ, $Q(R)$, with $R = knows \circ supervisor \circ colleague \circ married$ as follows.
$S_1 = \{knows \circ supervisor; supervisor \circ colleague \circ married\}$
$S_2 = \{knows \circ supervisor \circ colleague; colleague \circ married\}$
$S_3 = \{knows \circ supervisor; supervisor \circ colleague; colleague \circ married\}$
Step 2 For each set of subqueries S_i, $1 \leq i \leq N$, estimate the searching cost of S_i, which is defined as the maximum searching cost of the subqueries belong to S_i.
Step 3 Compare the searching cost of all sets of subqueries in S to find out the set S_i which has the minimum searching cost.

Note that, in *Step 1* above, we find all of the possible combinations of sequenced labels, and it takes polynomial time. Here, we consider only the labels as the split labels if it is not at the position of the labels with bounded Kleene operator or inside a bracket of an alternation operator. After finding the set of subqueries in *Step 3*, each subquery is evaluated on different CPU in parallel by using automata-based approach, and the results are gathered for the answer of the original RPQ. We will evaluate the efficiency of our proposed method applying on this problem in the next section.

Multi-query optimization of RPQs Consider a web-based public transportation system like Naver Maps, in which users issue queries in the form of RPQs to find optimal paths from their locations to the destinations. A large number of queries could be sent to the system at the same time. For such system, the batch of queries is expected to be evaluated efficiently in real-time manner. This problem is referred as the multi-query optimization. Recently, a framework, called SWARMGUIDE, has been proposed by Abul-Basher (2017) to optimize multiple regular path queries in graph databases. The framework detects commonalities among the RPQs by using a subgraph isomorphism technique, in order to find an optimal execution plan that is globally optimized over the plan spaces of the constituent RPQs. This approach only exploits the structure of RPQs, but does not exploit graph schema which could affect considerably the response time of RPQs evaluation as presented in the previous section. A simple idea to improve

Table 2 Summary of real-life datasets

Dataset	$\|V\|$	$\|E\|$	$\|Label\|$
Alibaba	52,050	340,775	649
Yago	1,756,958	3,615,249	13
Freebase	2,303,121	3,224,470	16

the performance of multi-query processing by using our proposed method is that (i) estimate the searching cost of each query; (ii) divide the batch of queries into k buckets (number of buckets depends on the system capacity) so that total estimated searching cost of a bucket is close to each other; (iii) evaluate the buckets in parallel fashion (e.g., multi-core CPUs or parallel on distributed systems).

6 Experimental evaluation

To evaluate the effectiveness of our proposed approach, we conducted two main experiments: the first one is to compare our estimated cost with the true cost which is the number of traversed edges during evaluation RPQs by using automata-based approach, and the other one is to compare our USCM-based approach with the automata-based approach (called AUT) (Goldman and Widom 1997) and the threshold rare label based approach (called TRL) (Koschmieder and Leser 2012) regarding the response time of parallel RPQs evaluation on large graphs.

6.1 Evaluation settings

Environments Our experiments were conducted on a personal computer which has 3.5 GHz Intel Core i3, 4 CPU cores, and 8.0 GB of RAM. All algorithms are implemented in Java.
Data and queries set We used three real-world graphs to verify the adaptability of the proposed method. The first one is from a research on biology (called Alibaba), the second one is called Yago dataset which is a semantic knowledge base, derived from Wikipedia, WordNet, and GeoNames (Suchanek et al. 2007), and the last one is Freebase dataset which provided by Bast et al. (2014). We summarized the properties of real-world datasets in Table 2. We also generated synthetic graphs with various graph size as well as average degree for the extensive evaluation. The details of dataset and queries set are as the followings.

We used Alibaba graph and the queries set given by previous research (Koschmieder and Leser 2012). The graph is a network of protein–protein interactions which is used regularly in biology systems, for instance, to discover protein functions and pathways in biological processes (Zahiri et al. 2013). This graph has 52,050 nodes, 340,775 edges, and 649 labels. We analyzed 10,000 queries in the queries set and found the following properties. The queries set has around 87% proportion of having simple RPQs, 3% proportion of having nested RPQs without recursive modifiers, and 10% proportion of having nested RPQs with recursive modifiers. For setting the queries set to be the same between our approach and the others, we replaced the modifiers (*, +, ?) by a fixed bounded recursion from 1 to 5.

Yago is a huge semantic knowledge base, extracted and combined entities and facts from Wikipedias in different languages. Currently, to provide knowledge based on the demand of users, Yago3 dataset (version 3 of Yago) is divided into specific portions, and each portion is called a theme (Mahdisoltani et al. 2013). For example, GEONAMES theme contains data of geographical entities and classes taken from GeoNames; meanwhile, CORE theme has main entities of yago and the facts between entities. To evaluate our proposed method, we extracted a knowledge graph from CORE them of Yago. This graph has 1,756,958 nodes, 3,615,249 edges, and 13 labels. Each node represents an entity such as a person, an organization, or a city; while, an edge represents the relationship between two entities, and it is assigned by a label as a fact (e.g., $hasChild$, $isLeaderOf$, $isLocatedIn$, etc.). We created a set of 30 queries with different length from 4 to 8, each has a sense in particular, which can be used for querying on Yago dataset to get some knowledge. For instance, the query with a regular expression R like

$$isMarriedTo \circ isPoliticianOf \circ isLocatedIn \circ hasCapital$$

is used to find out people who married to politicians living in the capital of a country. This queries set has 10 queries for each type of queries as we presented above. In which, the bounded Kleene operator has bounded recursion in the range from 1 to 5.

Freebase (Bollacker et al. 2008) is a large knowledge graph of the facts around the world, which is developed by Metaweb Technologies company in 2007 and was acquired by Google Inc. in 2010. This original dataset provides raw data dumped in RDF (Resource Description Framework) triples and it has several issues like name disambiguation and duplicate entities. We therefore sought to find a different version of Freebase dataset which had solved the known major issues of the original one. This new dataset is available to download from http://freebase-easy.cs.uni-freiburg.de/dump (Bast et al. 2014). In this dataset, each triple is a fact in the form of $<subject>$ $<predicate>$ $<object>$. It corresponds to an edge on the knowledge graph, in which a $<predicate>$ is considered as an edge-label. We extracted all triples which have predicates representing the relationship between two entities and ignored other ones. For instance, we used a triple $<Ann\ Taylor>$ $<Has\ Child>$ $<Christian\ Noel\ Davis>$, but did not use $<Ann\ Taylor>$ $<Weight>$ $<57.2>$. Finally, we obtained a Freebase graph with 2,303,121 nodes, 3,224,470 edges, and 16 labels. We also generated 30 queries with different length from 4 to 8 to evaluate our method on this dataset.

To gain much deeper understanding of our proposed method in the aspects of various parameters such as graph size or average degree of graph, we generated the synthetic graphs with various number of nodes and number of edges (for more details, see Tables 3 and 4) by using Gephi (Bastian et al. 2009). We used 15 distinct labels to annotate edges for these graphs. The occurrence of labels follows the Zipfian distribution. Then, we generated randomly 1000 RPQs with various lengths between 6 and 12. This queries set has about 5% proportion of having the alternation and 30% proportion of having the bounded Kleene operator with bounded recursion in the fixed range from 1 to 5.

Table 3 Accuracy evaluation with varied graph sizes

$\|V\|$	|E|	Average degree	Accuracy
2000	38,142	19	87.62
4000	76,860	19	88.43
8000	152,245	19	89.02
16,000	306,806	19	88.70
32,000	615,047	19	89.18
64,000	1,216,584	19	88.56

Table 4 Accuracy evaluation with varied average degrees

$\|V\|$	|E|	Average degree	Accuracy
16,000	63, 540	4	75.12
16,000	128, 297	8	81.52
16,000	255, 578	16	88.00
16,000	513, 225	32	89.06
16,000	1, 024, 183	64	88.45

Algorithms To evaluate the accuracy of our estimation method, we reimplemented AUT approach to measure the true cost, and measure the estimated cost by implementing our proposed method. For each query in the queries set, the closeness between the estimated cost, e_i, and the true cost, t_i, is calculated by the fraction $\frac{e_i}{t_i}$ in the case of $e_i \leq t_i$, otherwise, it is $\frac{t_i}{e_i}$.

We also reimplemented AUT approach and the threshold TRL approach to compare their response time with our USCM-based approach.

6.2 Experimental results

6.2.1 Accuracy of our estimation method

In order to evaluate the accuracy of our estimation method, we used three real-world datasets: Alibaba graph, Yago graph, Freebase graph, and a synthetic graph with 16,000 nodes and 306,806 edges. The queries set for each dataset is described in the previous subsection. The results show that our estimation method obtained high accuracy which is approximately 87% on average. Specifically, it is around 85%, 88%, 86%, and 89% in the case of Alibaba, Yago, Freebase, and the synthetic graph, respectively. As an example, Fig. 2 illustrates a comparison of the estimated cost and the true cost of 20 random queries on each dataset. We observed that the estimated cost is close to the true cost, excepts some queries with high cost caused by the bounded Kleene operators.

Next, we evaluate the accuracy with varied graph size (|V| + |E|). We generated synthetic graphs by varying the size and average degree of the graphs. In directed graphs, the average degree is defined by the fraction $\frac{|E|}{|V|}$. In the first case, we scale

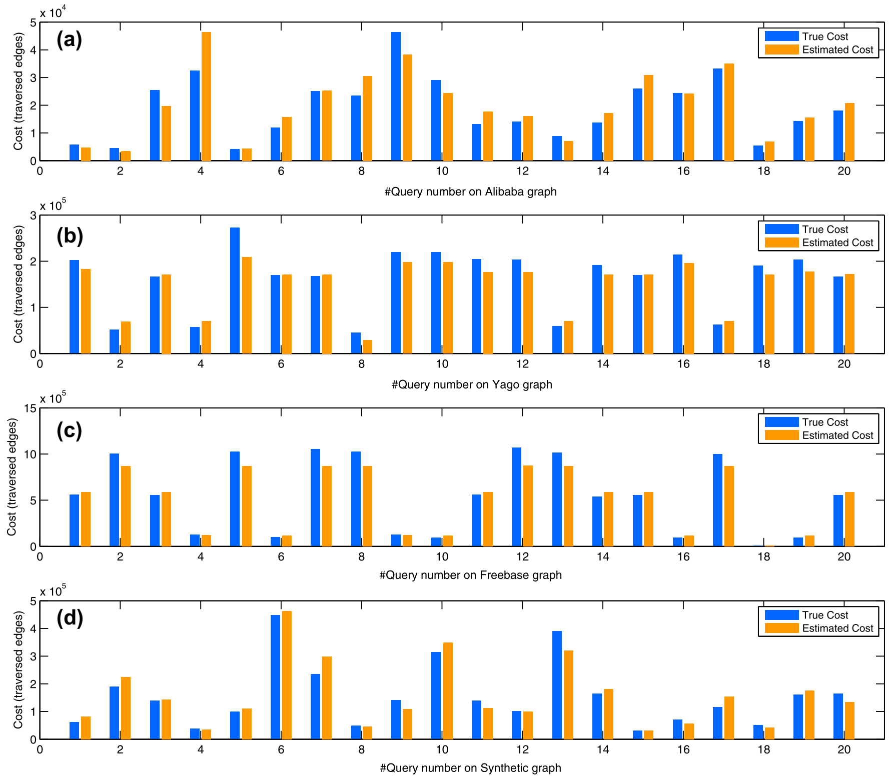


Fig. 2 Comparison of the true cost and the estimated cost. **a** #Query number on Alibaba graph, **b** #Query number on Yago graph, **c** #Query number on Freebase graph, **d** #Query number on Synthetic graph

graph size from around 40K to 640K nodes and edges, but we keep the same average degree for the graphs. As the results shown in Table 3, our estimation method obtained high accuracy at most 89% for all varied size of the graphs. In another case, we generated five synthetic graphs by fixing the number of nodes $|V| = 16$K and varying the number of edges $|E|$ from 64K to 1.0M. Consequently, the average degree of the graphs is varied from 4 to 64. We observed that the estimation accuracy for the graphs whose average degree is greater than or equal to 16, which is mostly around 89%, are higher than those in the cases of average degree of 4 and 8 (75.12% and 81.52%, respectively) as shown in Table 4. The results are reasonable because the higher average degree of the graph is, the higher probability of a label connected to anyone else, which helps to increase the estimation accuracy of our method, is.

Finally, we evaluate the impact of query path length on the estimation accuracy of our proposed method. To do this, we generate randomly 3000 queries (contain only concatenation operator) with varied length in the range from 3 to 8. Here, we do not need to evaluate the accuracy of estimating the queries with length equalling 2 because their true costs are the exact cost of unit-subqueries in USCM. Thus, we have 6 different subsets of queries, and each has 500 queries. We evaluated the queries on the synthetic graph with 16,000 of nodes and 306,806 of edges. Figure 3 shows that

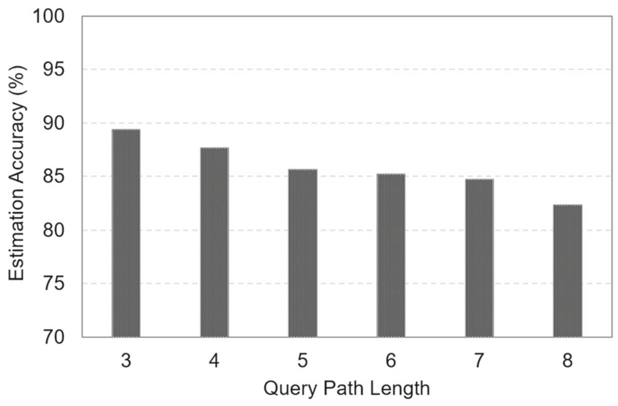


Fig. 3 Accuracy evaluation with varied query path length

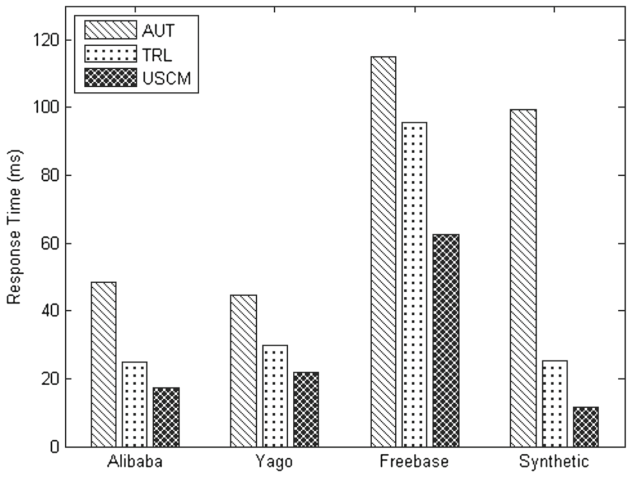


Fig. 4 Comparing the response time of parallel RPQs evaluation on large graphs

the accuracy decreases marginally from 89% (at query length equalling 3) to 82% (at query length equalling 8), which is still acceptable.

6.2.2 Efficiency of USCM-based parallel RPQs evaluation

We implemented our algorithm for parallel RPQs evaluation as described in Sect. 5. To ensure the parallel evaluation of split subqueries, the split subqueries are evaluated on different CPU and the results are gathered for the answer of an RPQ. To measure the response time, we get the timestamp difference between issuing an RPQ and getting the answer of an RPQ. That is, the response time of USCM-based approach includes the time for splitting the RPQ and combining partial answers. Figure 4 illustrates the average response times of three different approaches. We observed that our USCM-based parallel RPQs evaluation outperforms AUT. That is, using estimated cost of RPQ

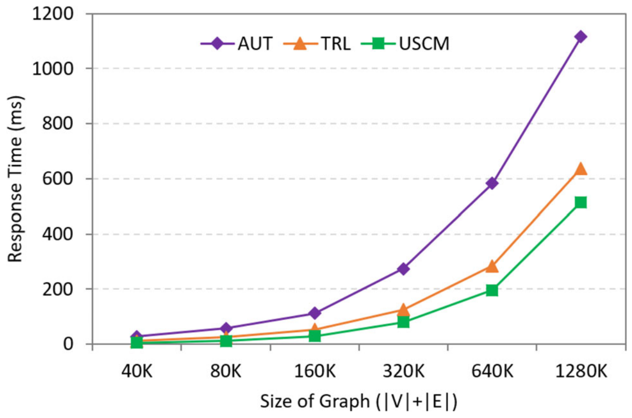


Fig. 5 Evaluating the response time of parallel answering RPQs with varied graph size

is necessary to reduce the searching cost of RPQs in parallel. We also observed that our approach reduced the average response time around 45%, 40%, 55%, and 120% comparing to TRL approach in case of Alibaba, Yago, Freebase, and the synthetic graph, respectively.

Figure 5 shows the comparison of the response time between our method and other methods with varied the size of the graph. The smallest graph has around 2K nodes and 38K edges, and the largest graph has 64K nodes and 1.2 million edges. As the result, the scaling of the implementation using the proposed method achieved a better performance with less response time than AUT and TRL approaches. Specifically, in the case of largest graph size, our method reduced the average response time around 120% and 24% comparing to AUT and TRL approaches, respectively.

6.2.3 Summary

From the experimental results, we find the following. (1) Our estimation method obtains high accuracy and scales well with the size of graphs. (2) The estimation accuracy on the graphs with a high average degree (e.g., greater or equals than 16) is higher than the ones having lower average degree. (3) Our proposed method is efficient when it is applied to parallel RPQs evaluation on large graphs.

7 Conclusions and future work

We proposed a novel approach for estimating the searching cost of RPQs on large graphs with cost functions based on the combinations of the searching cost of unit-subqueries. By exploiting unit-subqueries, we defined an Unit-Subquery Cost Matrix (USCM) which consists of all possible unit-subqueries of every RPQs. According to USCM, we provided cost functions for estimating the searching cost of a given RPQ. Experimental results illustrated the proposed method can estimate searching cost for

RPQs with high accuracy which is approximately 87% on average, and USCM-based approach outperforms traditional ones in the aspect of parallel RPQs evaluation.

We envision several directions of our work, one of them is extending the estimation of the searching cost for highly complex RPQs with respect to unbounded recursion. Besides, motivated by the absence of benchmarks devoted to RPQs, we want to develop a benchmark for estimating cost as well as evaluating RPQs. Moreover, we will focus on how to apply this approach for solving RPQ problems in various fields such as transportation analysis under disaster situations and social network analysis for recommendation systems.

Acknowledgements This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2016-0-00314) supervised by the IITP (Institute for Information & communications Technology Promotion). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2017R1A2B4012559).

References

Abul-Basher, Z.: Multiple-query optimization of regular path queries. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 1426–1430. IEEE (2017)

Almeida, J., Zeitoun, M.: Description and analysis of a bottom-up DFA minimization algorithm. Inf. Process. Lett. **107**(2), 52–59 (2008)

Barceló, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM Trans. Database Syst. **37**(4), 31 (2012)

Barceló Baeza, P.: Querying graph databases. In: Proceedings of the 32nd ACM SIGMOD–SIGACT–SIGAI Symposium on Principles of Database Systems, pp 175–188. ACM (2013)

Bast, H., Bäurle, F., Buchhold, B., Haußmann, E.: Easy access to the freebase dataset. In: Proceedings of the 23rd International Conference on World Wide Web. ACM, pp. 95–98 (2014)

Bastian, M., Heymann, S., Jacomy, M., et al.: Gephi: an open source software for exploring and manipulating networks. In: ICWSM, vol. 8, pp. 361–362 (2009)

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1247–1250. ACM (2008)

Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Rewriting of regular expressions and regular path queries. In: Proceedings of the Eighteenth ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems, pp. 194–204. ACM (1999)

Cong, G., Fan, W., Kementsietsidis, A.: Distributed query evaluation with performance guarantees. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 509–520. ACM (2007)

Consens, M.P., Mendelzon, A.O.: Graphlog: a visual formalism for real life recursion. In: Proceedings of the ninth ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems, pp. 404–416. ACM (1990)

Cruz, I.F., Mendelzon, A.O., Wood, P.T.: A graphical query language supporting recursion. In: ACM SIGMOD Record, vol. 16, pp. 323–330. ACM (1987)

Davoust, A., Esfandiari, B.: Processing regular path queries on arbitrarily distributed data. In: OTM Confederated International Conferences On the Move to Meaningful Internet Systems, pp. 844–861. Springer (2016)

Fan, W., Wang, X., Wu, Y.: Performance guarantees for distributed reachability queries. Proc. VLDB Endow. **5**(11), 1304–1316 (2012)

Fernandez, M., Suciu, D.: Optimizing regular path expressions using graph schemas. In: Proceedings, 14th International Conference on Data Engineering, 1998, pp. 14–23. IEEE (1998)

Fletcher, G.H., Peters, J., Poulovassilis, A.: Efficient regular path query evaluation using path indexes. In: Proceedings of the 19th International Conference on Extending Database Technology (EDBT), pp. 636–639 (2016)

Goldman, R., Widom, J.: Dataguides: enabling query formulation and optimization in semistructured databases. In: VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, 25–29 Aug 1997, Athens, Greece, pp. 436–445 (1997). http://www.vldb.org/conf/1997/P436.PDF. Accessed 23 Aug 2017

Grahne, G., Thomo, A.: An optimization technique for answering regular path queries. In: WebDB (Selected Papers), pp. 215–225. Springer (2000)

Grahne, G., Thomo, A.: Query containment and rewriting using views for regular path queries under constraints. In: Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 111–122. ACM (2003)

Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, (2006)

Konstas, I., Stathopoulos, V., Jose, J.M..: On social networks and collaborative recommendation. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 195–202. ACM (2009)

Koschmieder, A., Leser, U.: Regular path queries on large graphs. In: Scientific and Statistical Database Management, pp. 177–194. Springer, Berlin (2012)

Kossmann, D.: The state of the art in distributed query processing. ACM Comput. Surv. **32**(4), 422–469 (2000)

Le Anh, V., Kiss, A.: Efficient processing regular queries in shared-nothing parallel database systems using tree-and structural indexes. In: ADBIS Research Communications (2007)

Libkin, L., Vrgoč, D.: Regular path queries on graphs with data. In: Proceedings of the 15th International Conference on Database Theory, pp. 74–85. ACM (2012)

Liu, T., Liu, A.X., Shi, J., Sun, Y., Guo, L.: Towards fast and optimal grouping of regular expressions via DFA size estimation. IEEE J. Sel. Areas Commun. **32**(10), 1797–1809 (2014)

Liu, D., Huang, Z., Zhang, Y., Guo, X., Su, S.: Efficient deterministic finite automata minimization based on backward depth information. PloS ONE **11**(11), e0165864 (2016)

Mahdisoltani, F., Biega, J., Suchanek, FM.: Yago3: a knowledge base from multilingual Wikipedias. In: CIDR (2013)

Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. SIAM J. Comput. **24**(6), 1235–1258 (1995)

Nguyen-Van, Q., Tung, LD., Hu, Z.: Minimizing data transfers for regular reachability queries on distributed graphs. In: Proceedings of the Fourth Symposium on Information and Communication Technology, pp. 325–334. ACM (2013)

Scott, J., Ideker, T., Karp, R.M., Sharan, R.: Efficient algorithms for detecting signaling pathways in protein interaction networks. J. Comput. Biol. **13**(2), 133–144 (2006)

Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: Proceedings of the 16th International Conference on World Wide Web, pp. 697–706. ACM (2007)

Suciu, D.: Distributed query evaluation on semistructured data. ACM Trans. Database Syst. **27**(1), 1–62 (2002)

Trißl, S.: Cost-based optimization of graph queries. In: Proceedings of the SIGMOD/PODS PhD Workshop on Innovative Database Research (IDAR) (2007)

Trißl, S., Leser, U.: Estimating result size and execution times for graph queries. In: ADBIS (Local Proceedings), pp. 11–20 (2010)

Tung, L.D., Nguyen-Van, Q., Hu, Z.: Efficient query evaluation on distributed graphs with Hadoop environment. In: Proceedings of the Fourth Symposium on Information and Communication Technology, pp. 311–319. ACM (2013)

Yakovets, N., Godfrey, P., Gryz, J.: Query planning for evaluating SPARQL property paths. In: Proceedings of the 2016 International Conference on Management of Data, pp. 1875–1889. ACM (2016)

Yang, J., Leskovec, J.: Patterns of temporal variation in online media. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, pp. 177–186. ACM (2011)

Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2015)

Zahiri, J., Hannon Bozorgmehr, J., Masoudi-Nejad, A.: Computational prediction of protein–protein interaction networks: algorithms and resources. Curr. Genomics **14**(6), 397–414 (2013)